# CASE STUDY:
# Bookstore Management System

## Scenario

**Develop "Bookstore" application:** Write an application to manage a small bookstore selling three types of items: book, music CD/DVD, and software.

- Each Book contain 10 information: itemID, title, author, publisher, yearPublished, edition, volume, ISBN, price, quantity.
- Each MusicCD has 9 information: itemID, title, artist, publisher, yearPublished, volume, ISBN, price, quantity.
- Each Software has 8 information: itemID, title, publisher, yearPublished, version, ISBN, price, quantity.

**Task 1: The bookstore application allows the shop owner (or system admin) to do the following tasks:**

- List all items in the bookstore;
- Search an item in the bookstore;
- Add a new item: a newly published book OR a newly released CD/DVD or a newly released software ;
- Delete /Update an existing an item: the item may be a book or a CD/DVD or a software.

**Task 2: The bookstore application also handles a normal purchase transaction at the book store.**

A customer comes to the bookstore, select few items (book, music album, software) and make an order to buy them on the spot. The application allows the shop owner (or system admin) to process a purchase transaction as below:

- Ask customer their information to enter the system: name, address, phone number, email.
- Check out all the items in the customer's bucket: book, music album, software.
- Ask customer method of payment.
- The application then produces an invoice (based on above ordered items) including total cost, method of payment.

**Task 3: The application will be expanded to the online Bookstore system.**

The online bookstore system is web-based. The system will store all items (book, music CD/DVD, software) for sale and also store all customer information. As for an online transaction, a customer first logs in the bookstore system by entering a customerID and password. Then the customer can browse for titles or search by keyword. The customer puts some of the titles into a "shopping cart" which keeps track of the desired titles. When the customer is done shopping, he/she confirms the order, shipping address, and billing address. The bookstore system then issues a shipping order, bills the customer, and issues an electronic receipt. At the end of the transaction, the customer logs off.

**Online transaction scenario:**

- User logs on system → System validates the ID and password;
- User searches for a title by browsing or keyword search → User selects items to buy;
- System adds items to the customer's shopping card → System displays shopping cart, shipping address, and billing address;
- User confirms order and payment method;
- System processes order and produce an invoice.
- User logs off.

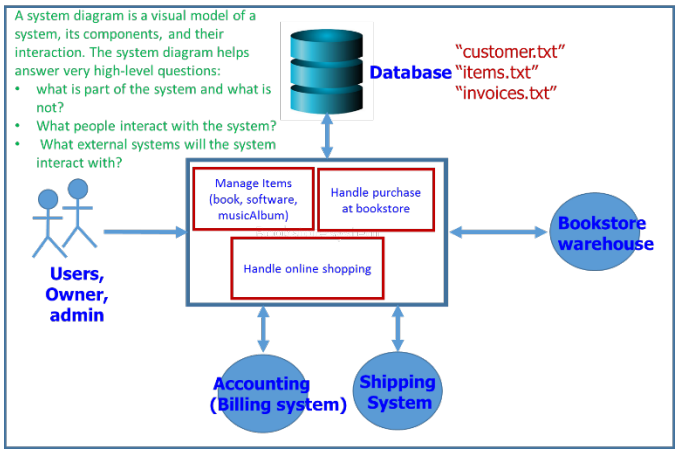
## System Diagram

*This helps answer very high-level questions: What is part of the system, and what is not? What people interact with the system? What external systems will our system interact with?*

A system diagram is a visual model of a system, its components, and their interaction. The system diagram helps answer very high-level questions:

- what is part of the system and what is not?
- What people interact with the system?

- What external systems will the system interact with?




# Candidate Classes & Class attributes

*Look for important nouns: Objects, people, organizations, places, events, concepts. Give them memorable names. Reflect real world objects where that is useful, and create new ones as needed to help support the scenarios.*

## Figure out candidate classes (class attributes & methods)

Find important nous: Object, people, organization, places, events, concepts. Give them memorable names. Reflect real world objects where that is useful, and create new ones as needed to help support the scenario.

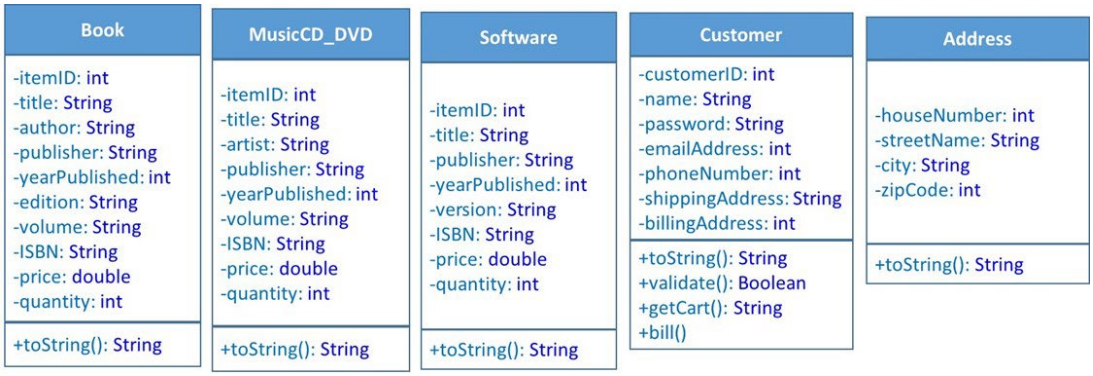
**+ In this application, the candidate classes are:**

- Bookstore; Customer; Book; Music Album, Software, Order, Address, ShoppingCart, …

**+ Identify class attributes: Analyze the system requirements to find the attributes of each class.**

**Note that some attributes may be references to other objects.**

- A Book has 10 information: itemID, title, author, publisher, yearPublished, edition, volume, ISBN, price, quantity;
- A Music CD/DVD has 9 information: itemID, title, artist, publisher, yearPublished, volume, ISBN, price, quantity;
- A software has 8 information: itemID, title, publisher, yearPublished, version, ISBN, price, quantity;
- A customer has these information: customerID, name, password, email address, phone number, shipping address, billing address. **Note that**: shipping address and billing address are referred to Address objects.
- An Address has these information: a house number, street name, city name, zip code;




## Relationships

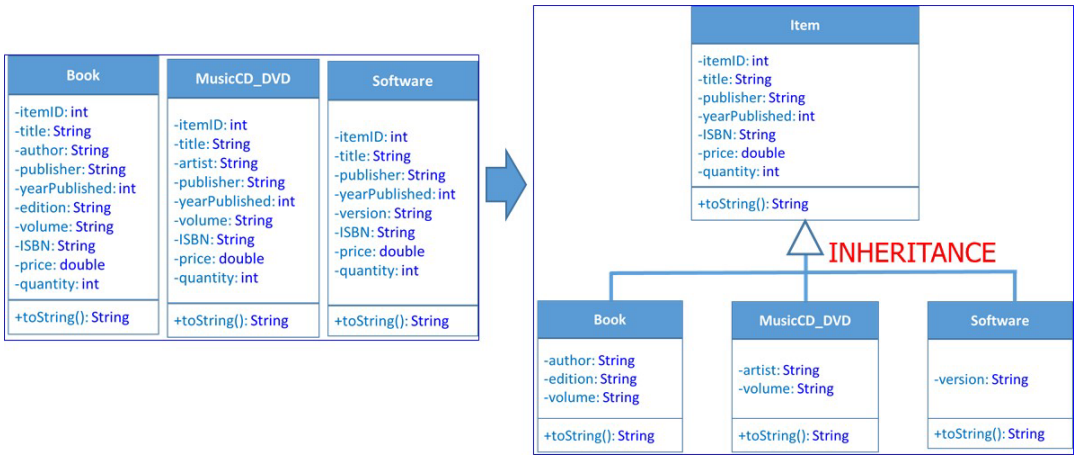
*Analyze the requirements to find relationships between classes. Draw relationships as lines between class boxes. Annotate the relationships: How many OrderItems for each Order are possible? Use symbols to indicate the kind of relationship: Generic, aggregation (has-a or owns-a), inheritance (is-a). For generic relationships, use an arrowhead to indicate whether the relationship is navigable: Does the source object know about the destination object?*

**+ After identify 3 classes Book, MusicCD_DVD, Software, we see there are commonalities between them:**

- *all these three classes have 7 common attributes (itemID, title, publisher, yearPublished, ISBN, price, quantity).*

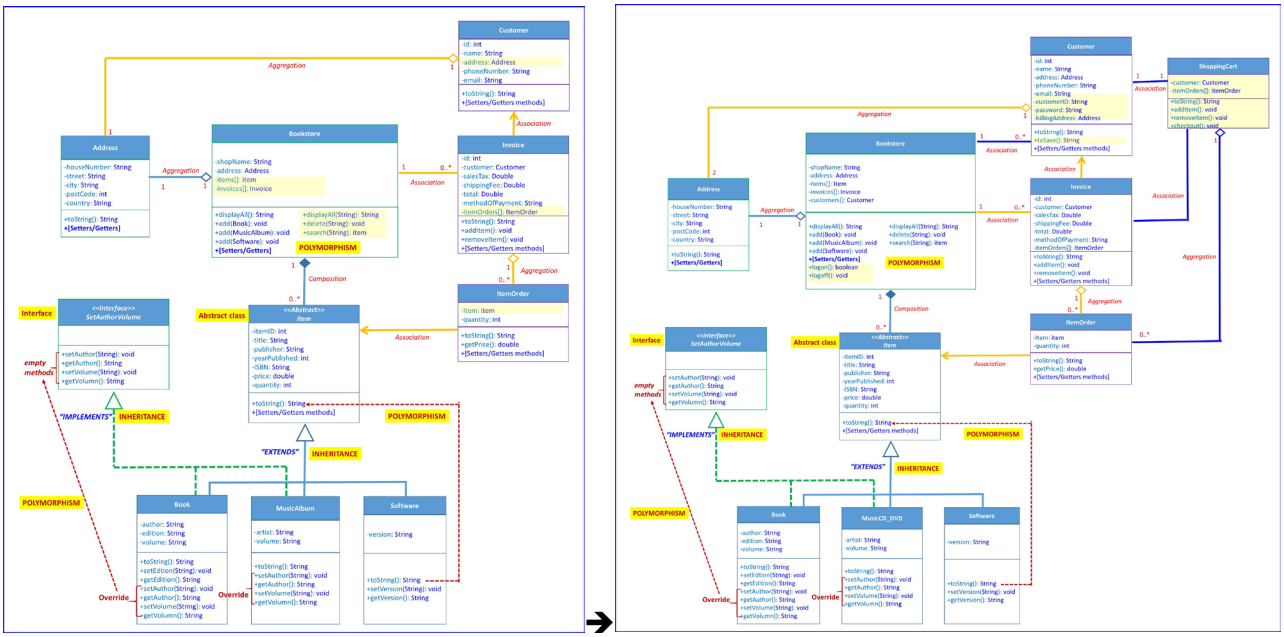
- *Therefore we'll factor these common attributes into a superclass, called "Item" and create INHERITANCE relationship.*
- *It can be seen also that: a book IS-A item, a MusicCD/DVD IS-A item, a Software IS-A item.*

**+ As a result, bellow is a class diagram of 4 classes (item, book, music CD/DVD, software) showing their INHERITANCE relationship:**




## Class Diagram

*Draw the entire system on one page (hide details about classes if you run out of room, and display them on subsequent pages). Be sure to include all important relationships, especially where one class depends on another.*



*Now go back and check the system against the scenarios; are all the scenarios supported by the system? Is all data owned by some class? Is it clear which class will be responsible for which actions?*

# Bookstore implementation (coding)

## Notes

**Method Names**

Note that the method names on the class diagram are very simple, usually just a verb. That's because the class itself usually supplies the noun: item.display(), customer.bill(). There's usually no reason to have methods like item.itemDisplay() or customer.customerBill(); they're redundant.

A special case arises when you want to extend a method of a superclass, as with Item.display(). For Book to override the superclass method, it _has_ to name it the same thing: Book.display(). That way, when clients call item.display(), they'll get the Book version if appropriate.

Inside Book.display(), it is usually advisable to start the method with a call to Item.display(); you can do this with the special syntax "super.display()". This executes the superclass version of the method, then returns to continue the subclass code.

In thinking about how this system would work in the real world, there are two critical pieces not shown on the diagram: **UI and data storage (database).** This is actually deliberate. The classes shown here form the "business logic" of the application, which should remain the same no matter what kind of UI or data storage mechanism you choose. You don't have to show everything on one diagram.

**Relationships:**

- A one-to-one or many-to-one relationship is often most easily expressed with a simple object reference, for example _shippingAddress.
- A one-to-many relationship usually needs some kind of collection of references, for example _itemOrders.
- A many-to-many relationship is difficult and best avoided if possible.

## Example Code

Customer.java and ShoppingCart.java are examples of translating a design to Java code. They compile except for undefined references to the missing classes. Customer.java has some examples of methods that provide controlled access to the customer object's data. Often these are just get/set pairs, but not always. The accessor methods aren't shown on the class diagram (although they could be). Often it's easiest to let them be implied by the variables shown on the diagram, unless you need to be very precise.

I like to name member variables starting with a leading underscore: _item, _address.

- This can help distinguish member variables from local variables and method parameter names.
- There are other ways of doing this as well.

```
class Customer {
    private String _name;              // Name of customer
    private final String _customerID;  // Logon ID
    private String _password;          // Logon password
    private Address _shippingAddress;  // Address to ship items to
    private Address  billingAddress;   // Billing address (if provided)

    // The name of a customer can be retrieved and set:
    public String getName() {return _name;}
    public void setName(String name) {_name = name;}

    // A customer ID can only be retrieved; it cannot be
    // changed:
    public String getCustomerID() {return _customerID;}

    // A password can only be set and validated with validate(),
    // not retrieved:
    public void setPassword(String password) {
        _password = password;
    }

    public Address getShippingAddress() { return _shippingAddress;}

    // If no billing address is supplied, it's the same as the
    // shipping address:
```

```
    public Address getBillingAddress() {
        if (_billingAddress != null)
            return _billingAddress;
        else
            return _shippingAddress;
    }

    // Check id and password to see if logon should succeed:
    public boolean validate(String id, String password) {
        return true;
    }

    // Access the customer's shopping cart:
    public ShoppingCart getCart() {return null;}

    // Bill the customer for a completed order:
    public void bill(Order order) {}
};

import java.util.*; // For Vector class

class ShoppingCart {
   private Vector _itemOrders;

   public void addItem(Item i) {}
   public void display() {}
   public void save() {}
};
```